REMARKS

Claims 1-48 are pending in the instant application. The Examiner has objected to claims 13 and 29 as comprising claim limitations that are lacking in antecedent basis. Claims 13-14 and 29-30 have been rejected under 35 USC 102(b) as allegedly being anticipated by U.S. Patent No. 5,854,929 to Van Praet et al., (hereinafter Van Praet). Additionally, claims 1-7,10-12, 15, 17-23, 26, 28-31, and 47 stand rejected under 35 U.S.C. 103(a) as allegedly being unpatentable over Van Praet, in view of U.S. Patent No. 6,477,683 to Killian et al., (hereinafter Killian), and further in view of U.S. Patent No. 5,428,792 to Conner et al., (hereinafter Conner). Also, claims 33-39 and 42-46 have been rejected under 35 USC 103(a) as allegedly being unpatentable over Van Praet. Furthermore, claims 8-9, 24-25, and 40-41 have been rejected under 35 U.S.C. 103(a) as allegedly being unpatentable over Van Praet, Killian, and Conner, as applied to claim 1 (for claims 8-9), claim 17 (for 24-25), and claim 33 (for 40-41); and further in view of U.S. Patent No. 6,003,038 to Chen, 6,003,038 (hereinafter Chen). Finally, Claims 16, 32, 40-41 and 48 have been rejected under 35 U.S.C. 103(a) as being unpatentable over Van Praet, and Conner; and further in view of Chen. Claims 13 and 29 have been amended to overcome the objections raised by the Examiner. Claims 1, 2, 5, 10, 13-15, 17, 18, 21, 26, 29-31, 33, 34, 37, 42, and 45-47 have been amended. The Applicant submits that the instant application is in condition for allowance for at least the reasons presented herein.

Rejections under 35 USC 102(b)

"A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. V. Union Oil Co. of California*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). Moreover, "[t]he identical invention must be shown in as complete detail as

is contained in the * * * claim." *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989).

To anticipate a claim under 35 U.S.C. § 102, a single source must contain all of the elements of the claim. *Lewmar Marine Inc. v. Barient, Inc.*, 827 F.2d 744, 747, 3 U.S.P.Q.2d 1766, 1768 (Fed. Cir. 1987), *cert. denied*, 484 U.S. 1007 (1988). Moreover, the single source must disclose all of the claimed elements "arranged as in the claim." *Structural Rubber Prods. Co. v. Park Rubber Co.*, 749 F.2d 707, 716, 223 U.S.P.Q. 1264, 1271 (Fed. Cir. 1984).

The Applicant's amended claims 13 and 29 recite a method and storage medium for compilation comprising: "generating a description of a computer architecture as a first library, the generating a description including describing software-visible physical objects of a computer as instances of classes, the classes comprised of methods for accessing and manipulating the software-visible physical objects, wherein the describing software-visible physical objects is performed using object-oriented classes, and at least one of the methods is described as implemented directly by instructions in an instruction set." Applicant's amended claims 14 and 30 recite a method and storage medium for re-targeting comprising "generating a first description of a first computer architecture as a first library, the generating a first description including describing software-visible physical objects and an instruction set of a computer as instances of classes, the classes comprised of methods for accessing and manipulating the software-visible physical objects, wherein the describing software-visible physical objects is performed using object-oriented classes, and at least one of the methods is described as implemented directly by instructions in the instruction set."

Van Praet does not recite these features. Specifically Van Praet does not teach or suggest the use of libraries to perform a method of compilation and re-targeting whereby a first library includes software-visible physical objects described as instances of classes in which the classes comprise methods for accessing and manipulating the software-visible physical objects. Rather, Van Praet recites a library format and a description

including vertices and storage elements (col. 5, lines 5-27; col. 18, lines 25-64) but does not teach or suggest the describing software-visible objects and instruction set of a computer as instances of classes, the classes comprising methods for accessing the manipulating the software-visible physical objects. Nor does Van Praet teach the describing software-visible physical objects using object-oriented classes. Because Van Praet does not teach or suggest each and every feature of Applicant's amended claims 13, 14, 29, and 30, the claims are not anticipated by Van Praet. Accordingly, the Applicant submits that claims 13, 14, 29 and 30 are in condition for allowance. Reconsideration of the outstanding rejections is respectfully requested.

Rejections under 35 USC 103(a)

For an obviousness rejection to be proper, the Examiner must meet the burden of establishing a *prima facie* case of obviousness. *In re Fine*, U.S.P.Q.2d 1596, 1598 (Fed. Cir. 1988). The Examiner must meet the burden of establishing that all elements of the invention are disclosed in the prior art; that the prior art relied upon, coupled with knowledge generally available in the art at the time of the invention, must contain some suggestion or incentive that would have motivated the skilled artisan to modify a reference or combined references; and that the proposed modification of the prior art must have had a reasonable expectation of success, determined from the vantage point of the skilled artisan at the time the invention was made. *In re Fine*, 5 U.S.P.Q.2d 1596, 1598 (Fed. Cir. 1988); *In re Wilson*, 165 U.S.P.Q. 494, 496 (C.C.P.A. 1970); *Amgen v. Chugai Pharmaceuticals Co.*, 927 U.S.P.Q.2d, 1016, 1023 (Fed. Cir. 1996).

1. Claims 1-7, 10-12, 15, 17-23, 26, 28-31, and 47

Applicant's amended claims 1 and 17 recite, respectively, a computer programming method and storage medium comprising: "describing data types as abstract data types without default or implicit implementations, the abstract data types comprising only non-concrete data types; distinguishing the abstract data types from classes or

interfaces, the classes or interfaces classified as either of abstract and concrete; and describing representations of values of the abstract data types as states of classes of objects with corresponding interfaces." None of the cited art references, alone or in combination, recites these features. Specifically, none of Van Praet, Killian, and Connor teaches or suggests describing data types as abstract data types without default or implicit implementation. Moreover, the references do not teach or suggest abstract data types being *only non-concrete data types*, and distinguishing the non-concrete abstract data types from classes that are classified as either abstract or concrete (emphasis added). Rather, Van Praet describes all data types as being concrete. Additionally, when taken in context, Van Praet suggests that 'concrete' is limited to that which is tied to storage (Van Praet, see generally Summary). In other words, the data type as taught by Van Praet is associated with a specified amount of storage for its data.

While the Examiner is correct in stating that the abstracting of data types is well known in metadata language specification, current programming languages do not use abstract data types as such without *also* allowing them to be made concrete. The Examiner has further suggested "[I]n case Van Praet does not already provide abstract data type to support the high-level model specification of the application, it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide abstract data type as taught by Killian because abstracting of data type enable further generation of subtypes thereby enhancing the process of abstracting the specification of the application model such as evidenced by object-oriented data type level of abstraction suggested by Killian and so well-known in most object-oriented modeling methodologies" (Office Action, p. 4-5). The Applicant respectfully disagrees. It would not have been obvious to combine the abstract data types because there would be no advantage in doing so without providing a second classifier (i.e., classes/interfaces) that is distinguished from the abstract data types in that the classes/interfaces are classified as "*either of abstract and concrete*" (emphasis added) as provided in the Applicant's amended claims 1 and 17. As recited in claims 1 and 17, the software visible physical objects that hold state in a computer are described using classes.

Concrete classes describe objects in computers, including both their structures and the methods by which their states may be altered. The physical objects are described as pre-existing global objects that may be said to represent the values of types by mapping each of their states to values of the types represented. Classes are written describing registers and main memory, and computer instructions may be represented as methods of those classes, resulting in a language that can be used as an assembly language for computer architecture.

Finally, the reference in Killian is to C++, a language in which both classes and types exist. However, according to Bjarne Stroustrup, the inventor of the C++ language, a "class is a user-defined data type" (Stroustrup, Bjarne. The Design and Evolution of C++. Reading, Mass.: 1994, p. 30). In other words, in the C++ language to which Killian makes reference, there is no difference between types and classes, such that types are always abstract. In C++, as in Van Praet, types are always concrete, and classes (which are nothing more than types defined by a programmer as opposed to types defined by the language definition) may be concrete or abstract.

The Applicant further submits that none of the cited art references teaches or suggests "describing representations of values of the abstract data types as states of classes of objects with corresponding interfaces." Van Praet's *classification* of "targeted instruction architecture into classes (e.g. Fig. 11)" is not the same as the use of an object-oriented programming language's feature called a *class*. Van Praet classifies instructions in Fig. 11 using a graph whose nodes are combined with "or-rules" and "and-rules". In contrast, in an object-oriented programming language, "class" is a textual description of an object involving no such rules. In an object-oriented programming language, a class specifies the construction of an object (e.g., what its components are and how its components are to be initialized), and the set of methods which may validly access and manipulate any object of the class. This is completely different from 'classification' *per se*.

Moreover, the Applicant submits that the Killian reference has been misapplied with respect to the Applicant's amended claims 1 and 17 because the C++ language

draws no distinction between classes and types with regard to their concreteness, as is recited in amended claims 1 and 17. Finally, the possible use of an object-oriented programming language to implement what is described by Killian and Van Praet, such as what is taught by Conner, does not use a programming language to describe *in that language* the software-visible hardware objects of which a computer is composed. Software-visible hardware objects are indeed described, but in data which may be processed by software written in object-oriented languages, not in the languages themselves. Moreover, it is well known in the art that the very design of the C++ and Java languages prevents the description of hardware as suggested by the Examiner.

Accordingly, the Applicant submits that amended claims 1 and 17 are not rendered obvious by the cited art references and that claims 1 and 17 are in condition for allowance.

Applicant's amended claims 2 and 18 recite respectively, "describing software-visible physical objects of a computer as instances of classes, the classes comprised of methods for accessing and manipulating the software-visible physical objects, wherein the describing software-visible physical objects is performed using object-oriented classes; and identifying when methods of the classes are implemented directly by the computer as instructions in an instruction set." None of the cited art references, alone or in combination, recites these features.

Amended claims 2 and 18 teach a language that enables a direct description of the software-visible hardware objects of a computer as instances of classes whose methods are instructions that can access and/or manipulate those objects. The Applicant submits that this is not the same as using a language to describe a computer. Rather, claims 2 and 18 teach using an *object-oriented* language to describe a computer *as objects* (emphasis added). The claim limitation "identifying when methods of the classes are implemented directly by the computer as instructions in an instruction set" enables a source code containing class method calls to be compiled such that some class method calls are

directly replaced by instructions in the instruction set of the target architecture, because the class is the class of a software-visible hardware object. In other words, the class is implemented directly by a computer. The source code also contains "ordinary" class method calls, where the classes are traditional object-oriented software classes. This mix of software and hardware classes in a source code is not taught or suggested by Van Praet, Killian, or Conner because these references do not apply to a programming language or source code. Unlike the present invention, Object-oriented languages such as C++ lack the features necessary to identify the hardware objects of a computer and differentiate them from software objects.

The Applicant submits that claims 2 and 18 are patentable over Van Praet, Killian, and Conner for the reasons provided above and due to their dependency on allowable claims 1 and 17, respectively. Reconsideration of the outstanding rejections is respectfully requested.

Turning now to claim 3 and 19, the cited references do not teach or suggest "describing multiple classes of pointer objects, said pointer objects capable of signifying objects" as the Examiner suggests. The language, C++ does not include the concept of multiple classes of pointer objects (i.e., C++ pointers do not have classes). Accordingly, the Applicant submits that the references are misapplied with respect to claims 3 and 19. It is submitted that claims 3 and 19 are in condition for allowance for the reasons described above and due to their dependencies on claims 1 and 17, respectively. The Applicant respectfully requests reconsideration of the outstanding rejections.

Referring to claims 4 and 20, the cited references do not teach or suggest "*describing commands* for transferring programs control in non-sequential ways *as routines*" (emphasis added) as suggested by the Examiner. While existing programming languages provide methods of transferring control between routines in non-sequential ways, Applicant's claims 4 and 20 teach a method of programming where these control

transfer statements are themselves designated routines. Thus, for at least this reason, and for the reasons that claims 4 and 20 depend from claims 1 and 17, respectively, the Applicant submits that claims 4 and 20 are not rendered obvious by the cited art references. The Applicant requests reconsideration of the outstanding rejections.

Applicant's amended claims 5 and 20 recite "describing interfaces, classes, enumerations, subroutines, and other routines as classes of objects." Claims 5 and 20 have been amended to better clarify that which the Applicant regards as the invention. The Applicant submits that the cited art references do not teach or suggest describing interfaces, classes, enumerations, subroutines and other routines as classes of objects. For at least this reason, and for the reason that claims 5 and 20 depend from allowable claims 1 and 17, respectively, claims 5 and 20 are in condition for allowance. Reconsideration of the rejections is respectfully requested.

With respect to Applicant's claims 6, 7, 22, and 23, none of the cited art references teach or suggest "invoking statements in a compilation with results thereof being incorporated into an output of a compiler...to interpret literals in an input of a compiler." Rather, Van Praet teaches the processing of statements in a compilation (col. 22, lines 61-64, col. 19, lines 13-15; col. 22, lines 15-60; col. 23, lines 2-28), but not the invoking of the statements being processed as recited in claims 6, 7, 22, and 23. For at least this reason and for the reason that claims 6, 7 and 22, 23 depend from allowable claims 1 and 17, respectively, the Applicant submits that claims 6, 7, 22, and 23 are patentable over the cited art references and are in condition for allowance. The Applicant respectfully requests reconsideration of the outstanding rejections.

The Applicant's amended claims 10 and 26 recite "describing a specification of formal arguments to routines as instances of objects, the objects' states representing the formal arguments required by the routines." Claims 11 and 27 recite "describing formal arguments to routines as a number of arguments

including type, interface, or class of each argument, dataflow attribute of each argument, and preconditions and postconditions of routines." Claims 10, 11, 26, and 27 are not obvious in view of the cited art references. The references do not teach or suggest specifying formal arguments to routines as instances of objects, whereby the objects' states represent the formal arguments required by the routines. The formal arguments represent the kinds of actual arguments that are required by routines. With regard to claims 11 and 27, the type, interface or class of an argument as defined in claims 1 and 17 are not equivalent to those taught by the cited art references. Moreover, while preconditions and postconditions are known in modeling, they have not been incorporated into any programming language except Eiffel, and not in combination with dataflow attributes, nor with the special distinction between type and class as recited in claims 1 and 17. Accordingly, claims 10, 11, 26, and 27 are not rendered obvious by the cited art references. For at least these reasons, and for the reason that claims 10, 11 and 26, 27 depend from claims 1 and 17, respectively, the Applicant submits that claims 10, 11, 26, and 27 are in condition for allowance. Reconsideration of the rejections is respectfully requested.

Turning to claims 12 and 28, the Applicant submits that claims 12 and 28 are patentable over the cited art references at least for the reason that they depend upon allowable claims 1 and 17, respectively. Reconsideration of the outstanding rejections is respectfully requested.

The Applicant submits that amended claims 15, 31, and 47 are in condition for allowance at least for the reasons provided above with respect to claim 2. Claims 29 and 30 are patentable over the cited art references for at least the reasons provided above with respect to claims 13 and 14, respectively. Reconsideration of the outstanding rejections is respectfully requested.

2. Claims 33-39 and 42-26

Claims 33-39 are patentable over Van Praet for at least the reasons provided

above with respect to claims 1-7 and 17-23. Claims 42-44 are patentable over Van Praet for at least the reasons provided above with respect to claims 10-12 and 26-28. Claims 45 and 46 are patentable over Van Praet at least for the reasons provided above with respect to claims 13, 29 and 14,30 respectively. The Applicant requests reconsideration of the outstanding rejections.

3. Claims 8-9, 24-25, and 40-41

The Applicant submits that claims 8, 24, and 40 are not rendered obvious by the cited art references. The terminology recited in claims 8, 24, and 40, namely "constant class" and "variable class" as used in claims 8, 24, and 40 describe the variability of instances of the classes, not the classes themselves. In other words, "constant class" refers to "class describing objects whose states cannot be modified" and "variable class" refers to "class describing objects whose states can be modified." In object-oriented programming languages such as C++, it is common to apply a "const" modifier to a subset of the methods defined in a class. This defines two versions of one class—a constant version and a variable version—and there are many language rules that apply to when objects defined with the constant version may or must be used, versus when objects defined with the variable version may or must be used. Claims 8, 24, and 40 teach that these two classes can be described through derivation: the variable class is derived from the const class. Then, the rules about when a "const" or "variable" object may or must appear become exactly those rules about when an object of a derived class may be used when an object of a base class is called for. Claims 8, 24, and 40 treat const and variable classes as base and derived, respectively, where variable classes are derived from const classes. While using a single descriptor may not itself be novel, the Applicant submits that using the single descriptor to define two classes, where one is derived from the other, is novel as provided in claims 8, 24, and 40. This derivation of variable classes from constant classes is not taught or suggested by Van Praet, Killian, or Conner, nor is it a mechanism used in any object-oriented languages referenced by the cited art.

Chen teaches a constant pool (col. 6, lines 9-35; FIG. 4); however, this constant pool is not used to store constant classes independently of variable classes. It is used to store constants, including classes—that is, class descriptors, where such descriptors are all constant (col. 6, lines 9-35). Additionally, the Chen reference applies to the implementation of a Java Virtual Machine, and the Java language does not enable a single class to be used to describe objects that may be modified and objects that may not be modified. Accordingly, the Applicant submits that Chen is misapplied with respect to claims 8, 24, and 40. Thus, claims 8, 24, and 40 are patentable over the cited references for at least the reasons provided above and for the reason that claims 8, 24, and 40 depend from allowable claims 1, 17, and 33, respectively. Reconsideration of the outstanding rejections is requested.

Claims 9, 25, and 41 mirror claims 8, 24, and 40, respectively, except that they apply to the interfaces of classes. Accordingly, claims 9 and 24 are patentable over the cited art references for at least the reasons stated above with respect to claims 8, 24, and 40 and because of their dependency on allowable claims 1, 17, and 33, respectively. Reconsideration of the rejections is respectfully requested.

4. Claims 16, 32, 40-41, and 48

Claims 16, 32, 40, and 48 are patentable over the cited art references at least for the reasons provided above with respect to claim 8. Claim 41 is patentable over the cited art references for at least the reasons provided above with respect to claims 9 and 25. Reconsideration of these rejections is respectfully requested.

In view of the foregoing, it is respectfully submitted that the instant application is in condition for allowance. Accordingly, it is respectfully requested that this application be allowed and a Notice of Allowance issued. If the Examiner believes that a telephone conference with Applicant's attorneys would be advantageous to the disposition of this case, the Examiner is cordially requested to telephone the undersigned.

No new matter has been entered and no additional fees are believed to be required. In the event the Commissioner of Patents and Trademarks deems additional fees to be due in connection with this application, Applicant's attorney hereby authorizes that such fee be charged to Deposit Account No. 06-1130 maintained by Applicant's attorneys.

Respectfully submitted,

CANTOR COLBURN LLP

By _____
Marisa J. Dubuc
Registration No. 46,673
Customer No. 23413

Date: June 28, 2004
Address: 55 Griffin Road South, Bloomfield, CT 06002
Telephone: (860) 286-2929